Tries, Patricia Tries, and Suffix Trees

[Notes for the Training Camp]

Yufei Tao

ITEE
University of Queensland

In this lecture, we will look at data structures on a new type of elements: text strings. Our ultimate goal is solve a difficult problem called substring matching with a clever structure called the suffix tree.

We will get to the suffix tree in an incremental manner, first starting with the trie, and then progressing to its space-economical variant: the Patricia trie. Both of these more fundamental structures tackle a special version of the substring matching problem: exact matching.

Exact Matching

Define a string to be a sequence of characters, all of which are chosen from an alphabet Σ. Let S be a set of strings, each of which has a unique integer id. Given a string q, an exact matching query reports:

- the id of q if it exists in S
- nothing otherwise.

Example

Suppose that $S = \{$aaabb, aab, aabaa, aabab, aba, abbb, abbba, abbbb$\}$. Let the ids of these strings be (from left to right) 1, 2, ..., 8, respectively. Given $q =$ aabaa, a query returns id 3, whereas given $q =$ abab, it returns nothing.

Prefixes

Let s be a string of length $t = |s|$ (i.e., $|s|$ represents the number of characters in s). We can write its characters (from left to right) as $s[1]s[2], ...s[t]$, respectively.

For any $i \in [1, t]$, the string $s[1]s[2]...s[i]$ is called a prefix of s. Specially, an empty string $\emptyset$ is also a prefix of s.

Example

$s =$ aabaa has 6 prefixes: $\emptyset$, a, aa, aab, aaba, and aabaa.

Prefix-Free

A set S of strings is called prefix-free if no string in S is a prefix of any other string in S. Any set of strings can be made prefix-free by appending a special "termination symbol" to each string in S.

Example: Let $S = \{\texttt{aaabb}, \texttt{aab}, \texttt{aabaa}, \texttt{aabab}, \texttt{aba}, \texttt{abbb}, \texttt{abbba}, \texttt{abbbb}\}$. We can convert S to $\{\texttt{aaabb}\bot, \texttt{aab}\bot, \texttt{aabaa}\bot, \texttt{aabab}\bot, \texttt{aba}\bot, \texttt{abbb}\bot, \texttt{abbba}\bot, \texttt{abbbb}\bot\}$, which is prefix-free.

From now on, we will consider that S is prefix-free, and that every string in S ends with $\bot$.

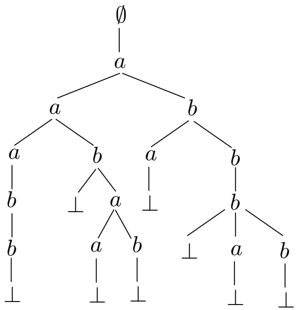
Tries

The trie on S is a tree T defined as follows:

- Each node u of T corresponds to a distinct string $P(u)$ which is a prefix of some string in S.
- Let u be a node, and v a child node of u. Then:
    - $P(u)$ is a prefix of $P(v)$.
    - $|P(v)| = |P(u)| + 1$.
- Each node u is labeled with a character c, which is the last character of $P(u)$.
- Each leaf z has its $P(z)$ equal to a string in S.

Example

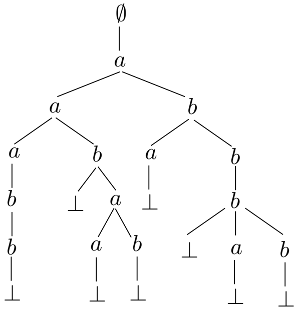
Let $S = \{$aaabb$\perp$, aab$\perp$, aabaa$\perp$, aabab$\perp$, aba$\perp$, abbb$\perp$, abbba$\perp$, abbbb$\perp\}$. The trie is:



Intuitively, a trie is a tree where strings share prefixes as much as possible.

Query

A trie answers an exact matching query q in $O(1 + |q|)$ time (think: how to use only $O(1)$ time to navigate to a child?).



How do we answer an exact matching query with $q =$ aabaa? How about $q =$ abab?

The number of nodes is $O(n)$, where n is the total length of all the strings in S, namely, $n = \sum_{s \in S} |s|$.

Let m be the number of strings in S. Note that n can be far larger than m.

Next, we will improve the space consumption to $O(m)$, without affecting the query time, provided that every string in S has been stored as an array. The new structure is called the Patricia trie.

Storing the Strings as Arrays

Henceforth, we will denote the strings in S as $s_1, s_2, ..., s_m$, respectively. We will consider that each s_i is stored in an array of size $|s_i|$, where $s_i[j]$ gives the j-th $(1 \leq j \leq |s_i|)$ character of s_i.

LCS

The longest common prefix (LCS) of a set S of strings is a string σ such that:

- σ is a prefix of every string in S.
- There is no string σ' such that σ' is a prefix of every string in S, and $|\sigma'| > |\sigma|$.

For example, the LCS of $\{\texttt{aaabb}\bot, \texttt{aab}\bot, \texttt{aabaa}\bot\}$ is aa, and that of $\{\texttt{aaabb}\bot, \texttt{baa}\bot\}$ is $\emptyset$.

Extension Set

Given two strings s_1, s_2, we use $s_1 \cdot s_2$ to denote their concatenation.

Let S be a set of strings, and σ the LCS of S. The extension set of S is the set of characters c such that $\sigma \cdot c$ is a prefix of at least one string in S.

Example

For example, the extension set of $\{\mathtt{aaabb}\bot, \mathtt{aab}\bot, \mathtt{aabaa}\bot\}$ is $\{\mathtt{a}, \mathtt{b}\}$.
The extension set of $\{\mathtt{aaabb}\bot, \mathtt{baa}\bot\}$ is also $\{\mathtt{a}, \mathtt{b}\}$.

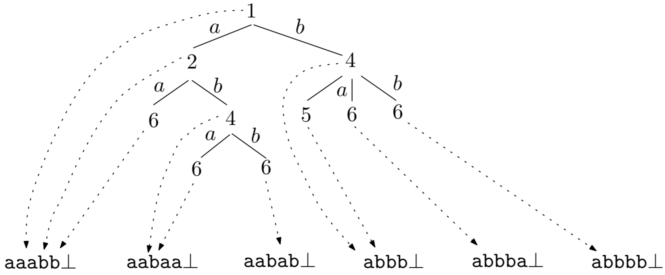
Patricia Trie

The Patricia trie T on S is a tree where each node u carries a positional index $PI(u)$, and a representative pointer $RP(u)$. T can be recursively defined as follows:

1. If $|S| = 1$, then T has only one node u with $PI(u) = 1$, and $RP(u)$ referencing the array of the (only) string in S.
2. Otherwise:
   - Let σ be the LCS of S. The root of T is a node u with $PI(u) = |\sigma|$, and $RP(u)$ referencing the array of an arbitrary string in S.
   - Let E be the extension set of S. Then, u has $|E|$ subtrees, one for each character c in E. Specifically, the subtree for c is a Patricia trie on the set of strings in S with $\sigma \cdot c$ as a prefix.

Example

Let $S = \{$aaabb⊥, aabaa⊥, aabab⊥, abbb⊥, abbba⊥, abbbb⊥$\}$. The Patricia trie of S is:

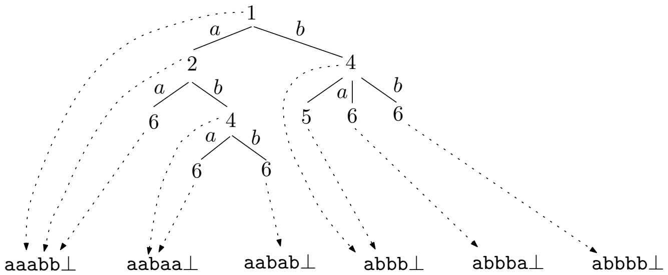


Lemma: A Patricia trie on m strings has at most $2m - 1$ nodes.

Proof: The Patricia trie has m leaves (each corresponding to a different string in S), and is a full binary tree (each internal node has 2 child nodes). □

Query

A Patricial trie answers a query with string q in $O(1 + |q|)$ time.



How would you answer an exact matching query with $q =$ aabab⊥. How about $q =$ abbab⊥?

The Prefix Matching Problem

Let S be a set of m strings, each of which has an integer id. Given a string q, a query reports the ids of all the strings $s \in S$ such that q is a prefix of s.

Example

Let $S = \{\texttt{abbba}\bot, \texttt{aabaa}\bot, \texttt{aaabb}\bot, \texttt{abbb}\bot, \texttt{aabab}\bot, \texttt{abbbb}\bot\}$, where the strings have ids 1, 2, ..., 6, respectively. Then:

- for $q = \texttt{ab}$, we should return ids 1, 4, 6.
- for $q = \texttt{aab}$, return 2, 5.
- for $q = \texttt{ba}$, return nothing.

The Prefix Matching Problem

Using a Patricia trie of $O(m)$ space, we can answer a prefix matching query with string q in $O(1 + |q| + k)$ time, where k is the number of ids reported.

Think: how?

The Substring Matching Problem

Let σ be a string of n characters. Given a string q, a query returns the starting positions of all the occurrences of q in σ.

Example

Let $\sigma = \texttt{aabbabab}$. Then:

- for $q = \texttt{abb}$, return 2 because substring abb starts at the 2nd position of σ.
- for $q = \texttt{bab}$, return 4 and 6.
- for $q = \texttt{bbb}$, return nothing.

Suffixes

For a string $s = s[1]s[2]...s[l]$, the string $s[i]s[i+1]...s[l]$ is called a suffix of s for each $i \in [1, l]$.

Clearly, a string s has $|s|$ suffixes.

Example

String aabbabab has 8 suffixes: aabbabab, abbabab, bbabab, babab, abab, bab, ab, b.

Suffixes

Recall that, in our substring matching problem, the input string is σ. Denote by S the set of all the suffixes of σ.

> A query string q is a substring of σ if and only if q is a prefix of a string in S.

Earlier, we proved the following for the prefix matching problem:

Lemma: Let S be a set of m strings, each of which has an integer id, and has been stored as an array. We can build a structure of $O(m)$ space such that, given a query string q, the ids of all strings $s \in S$ such that q is a prefix of s can be reported in $O(|q| + k)$ time, where k is the number of ids reported.

Suffix Trees

We thus immediately obtain:

Lemma: For the substring matching problem, we can build a structure of $O(n)$ space such that, given a search string q, a query can be answered in $O(1 + |q| + k)$ time, where k is the number of reported positions.

> The structure implied by the above lemma is called the suffix tree, which is essentially a patricia trie on all the suffixes of σ.